Improving MinBERT: Gradient Surgery and Mixed-Precision Training

Stanford CS224N Default Project

Maxwell Chen
Department of Computer Science
Stanford University
maxhchen@stanford.edu

Abstract

This aim of this project is to implement parts of the transformer model BERT (in particular, a lite-version called "minBERT") and train it to perform a number of downstream language tasks in tandem (multitask classification) including sentiment analysis of movie reviews. To extend my model performance, I performed a randomized hyperparameter sweep, incorporated "gradient surgery" techniques to address conflicting task gradients, performed mixed-precision training for model speedup, and model ensemble methods to combine per-task finetuned models for an "ultimate" model. In the end, I was surprised to find that gradient surgery contributed little to my model's overall performance, and saw that mixed-precision training led to both faster training and better accuracy, at the cost of numerical instability. Ultimately, the ensemble method combining multiple finetuned models led to the best outcome that I was able to achieve.

1 Key Information to include

- Mentor: Xiaoyuan Ni
- External Collaborators (if you have any): N/A
- Sharing project: N/A

2 Introduction

Over the last few years, it has become increasingly clear that we are experiencing a fundamental shift in the capabilities of machine learning models in light of products such as ChatGPT, DALL-E, and AlphaFold. These can all be considered Foundation Models or "Large Language Models" (despite being applied in domains not exclusive to language) – a class of models trained on vast amounts of data. These models can be traced back to earlier work on Transformer models, which leverage an attention mechanism to selectively weigh different things during training. As models grow increasingly complex, one direction of interest is the difficulty in getting models to achieve high performance on a variety of tasks at the same time. Various problems such as distribution shift or imbalance, and general difficulty in optimizing joint loss functions, have made multitask learning a difficult problem to solve, but there is continued interest in this area in the hopes of developing a general model (and ultimately, an artificial general intelligence, aka, "AGI") that can faithfully accomplish a broad spectrum of tasks, closer to what humans are capable of. In this report, I take MinBERT – a lightweight version of the classic BERT transformer model – and apply different techniques to improve its accuracy jointly across three natural language classification tasks further described below.

3 Related Work

This section helps the reader understand the research context of your work, by providing an overview of existing work in the area.

3.1 MinBERT and Attention

BERT (Devlin et al. (2018)), aka "Bidirectional Encoder Representations from Transformers", is a popular transformer model released in 2018 that became very popular as a backbone for researching transformer models that eventually moved towards LLMs and Foundational Models.

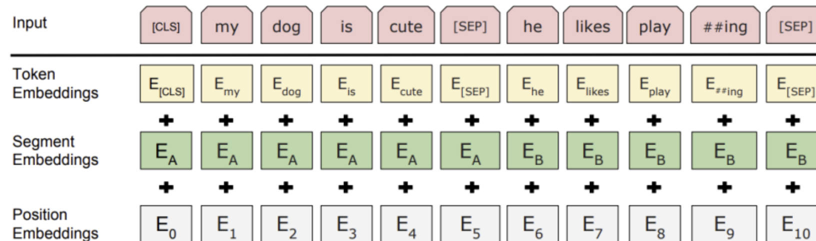
At its core, BERT tokenizes sentences into words, into individual "word pieces" in its internal dictionary; unknown words are assigned [UNK] tokens, and special tokens are used to denote sentence start (e.g. [CLS]) and sentence separation (e.g. [SEP]).

Figure 1: Example of BERT producing semantic tokens.

Word	Word Pieces
snow	[snow]
snowing	[snow, ##ing]
fight	[fight]
fighting	[fight,##ing]
snowboard	[snow,##board]

These tokens are passed through an embedding layer, and combined with segment (which sentence) and position (word placement within sentence) embeddings to produce the final embeddings.

Figure 2: Example combination of multiple embeddings.



These are then passed through a series of transformer layers utilizing multi-head self-attention, feedforward layers with ReLU activations, and dropout.

3.2 Gradient Surgery

In "Gradient Surgery for Multi-Task Learning" (Yu (2020)), the authors acknowledge that training on multiple tasks jointly creates a complicated optimization that is often hard to untangle and risks compromising the model's overall performance instead of improving task-specific performance. One issue proposed as being "central" to this difficulty is the notion of "conflicting gradients" – gradients for different tasks that have a negative cosine similarity, meaning they point in different directions and impede overall training by contributing reduced combined progress in the gestalt optimization landscape.

One method I experimented with was the proposed "gradient surgery" algorithm titled "PCGrad", standing for "projecting conflicting gradients." This approach alters conflicting gradients by projecting each one onto the normal plane of the other, ensuring that they do not impede movement in the optimization landscape on the account of the other.

Figure 3: PCGrad Algorithm Pseudocode.

Algorithm 1 PCGrad Update Rule

Require: Model parameters θ, task minibatch $\mathcal{B} = \{\mathcal{T}_k\}$

1: $\mathbf{g}_k \leftarrow \nabla_\theta \mathcal{L}_k(\theta) \;\; \forall k$

2: $\mathbf{g}_k^{\text{PC}} \leftarrow \mathbf{g}_k \;\; \forall k$

3: **for** $\mathcal{T}_i \in \mathcal{B}$ **do**

4: **for** $\mathcal{T}_j \overset{\text{uniformly}}{\sim} \mathcal{B} \setminus \mathcal{T}_i$ in random order **do**

5: **if** $\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j < 0$ **then**

6: *// Subtract the projection of* $\mathbf{g}_i^{\text{PC}}$ *onto* $\mathbf{g}_j$

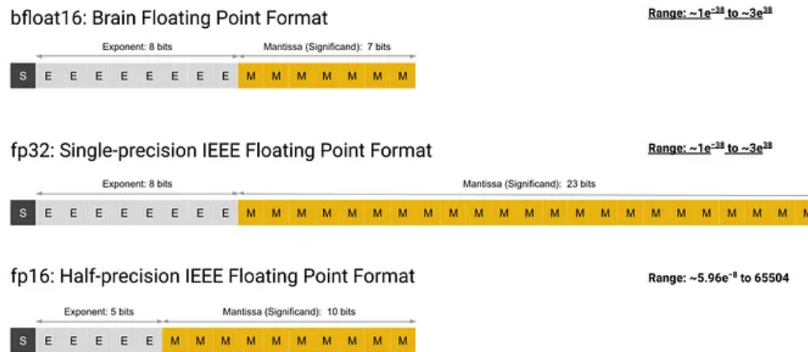
7: Set $\mathbf{g}_i^{\text{PC}} = \mathbf{g}_i^{\text{PC}} - \frac{\mathbf{g}_i^{\text{PC}} \cdot \mathbf{g}_j}{\|\mathbf{g}_j\|^2} \mathbf{g}_j$

8: **return** update $\Delta\theta = \mathbf{g}^{\text{PC}} = \sum_i \mathbf{g}_i^{\text{PC}}$

3.3 Mixed-Precision Training

As researchers seek to push out more and more efficiency from their training pipelines, hardware issues – caching, network I/O, memory, etc. – become increasingly relevant. PyTorch is a popular library used for machine learning development, and has functionality for advanced hardware profiling and optimization. For instance, PyTorch is capable of making models use Mixed-Precision Training (Micikevicius et al. (2017)), a term for allowing certain values to use lower-precision datatypes. For instance, instead of regular FP32 values, we can make a model use FP16 values where possible, or "BFloat16" values if compatible (representing FP16 values that still use the full dynamic range of FP32), meaning that our model can train much faster at the risk of precision and numerical instability.

Figure 4: Explanation of Mixed-Precision Training and Data Types.



4 Approach

The fundamental architecture for my approach was the completed MinBERT model for which the skeleton was provided by course staff. I then added a simple fully-connected layer for each of the three task heads to the desired output shape to produce logits when performing loss function calculations.

4.1 Multitask Classification

I modified ‘multitask_classifier.py‘ to take batches from each dataset at the same time in order to use a "round robin" approach where the model ideally learns all three tasks uniformly.

After computing the loss for each individual task, I then combined the losses and performed back-propagation to update my model from a single batch step across all three tasks.

4.2 Model Configuration and Hyperparameter Sweep

I performed a hyperparameter sweep with the help of Weights & Biases, a popular platform and tool for managing and logging machine learning and data science experiments. Details of the sweep are presented in "Experimental Details" and "Results."

4.3 Novelty Extensions

Using the best set of parameters found (described later in greater detail), I applied the two techniques I was interested in evaluating as discussed earlier in "Approach": **(1) Gradient Surgery vis-a-vis "PCGrad"**, and **(2) PyTorch Lightning for speedup and Mixed-Precision Training w/ FP16** to see whether they improved classification accuracy or training time.

4.4 PCGrad

I originally implemented PCGrad on my own, but was unsure if I had implemented it correctly due to the underwhelming results (discussed later). I tried using a preexisting implementation (Tseng (2021)) of PCGrad and more or less got the same results.

4.5 Ensemble Models

Finally, I took the best model settings I found from the experiments above, trained for an extended period (20 epochs), and created three different copies of the model. I performed finetuning on each model solely on the dataset for a single task, and then created an ensemble model where we use each respective submodel’s prediction for a datapoint for its corresponding task.

5 Experiments

5.1 Data

The three tasks of interest are **(1) sentiment analysis**, **(2) paraphrase detection**, and **(3) semantic textual similarity**.

5.1.1 Sentiment Analysis

Sentiment analysis, hence referred to as SST for simplicity, seeks to classify the polarity of a piece of text. The provided datasets being used for training and evaluation are the **SST and CFIMDB datasets**. For SST, the dataset originally consists of 215,154 phrases parsed from sentences in movie reviews with varying sentiment labels on a scale from 1 to 5. We are provided the following splits:

- train = 8,544 examples
- dev = 1,101 examples
- test = 2,210 examples

For CFIMDB, the dataset originally consists 2,434 highly polar movie reviews with binary sentiment labels ("positive" or "negative). We are provided the following splits:

- train = 1,701 examples
- dev = 245 examples
- test = 488 examples

For multitask classification, we focus on the SST dataset for simplicity.

5.1.2 Paraphrase Detection

The dataset used for paraphrase detection is the **Quora dataset**. It originally consists of 400,000 labeled question pairs, which was broken into the following provided splits:

- train = 141,506 examples
- dev = 20,215 examples
- test = 40,431 examples

Each label identifies whether the questions in each pair are paraphrases of one another.

5.1.3 Semantic Textual Similarity

The dataset for Semantic Textual Similarity, hence referred to as STS, is the **SemEval dataset**. It originally consists of 8,628 sentence pairs with varying similarity labels, which are split into the following subsets:

- train = 6,041 examples
- dev = 864 examples
- test = 1,726 examples

Each label describes whether the pair of sentences have the same meaning semantically (score of 5), or are completely unrelated (score of 0).

5.2 Evaluation method

For the SST/CFIMDB datasets, the evaluation metric will be accuracy on the dev/test split between the predicted sentiment label and the true sentiment label.

We are given baseline expectations for the SST/CFIMDB datasets with:

- Pretraining for SST: Dev Accuracy: 0.390 (0.007)
- Pretraining for CFIMDB: Dev Accuracy: 0.780 (0.002)
- Finetuning for SST: Dev Accuracy: 0.515 (0.004)
- Finetuning for CFIMDB: Dev Accuracy: 0.966 (0.007)

For the Quora dataset, the evaluation metric will be accuracy on the dev/test split between the predicted pair label and the true pair label.

For the SemEval dataset, the evaluation metric will be the Perason correlation of the true similarity label against the predicted similarity label.

For the Quora and SemEval datasets, we can evaluate our model's performance against other groups using the class leaderboard since we don't have a rough baseline.

Additionally, we are provided with F1 scores after performing evaluation on the multitask classifier's overall performance.

When debugging, I tried doing a qualitative evaluation by looking at specific mislabeled examples and trying to identify linguistic or semantic reasons for what may have caused the model to mislabel (e.g. wrong synonym or meaning, negations, etc.), but this proved to be very difficult.

5.3 Experimental details

The hyperparameter sweep mentioned in "Approach" was performed over parameter values in the following ranges: **Training Mode** [Pretrain, Finetune], **Learning Rate** [1e-3, 1e-4, 1e-5], **Batch Size** [8, 32, 64], and **Dropout Probability** [0.2, 0.3, 0.4].

This randomized search was performed over 5 epochs to get a rough sense of which parameter sets led to faster convergence before performing further experimentation.

5.4 Results

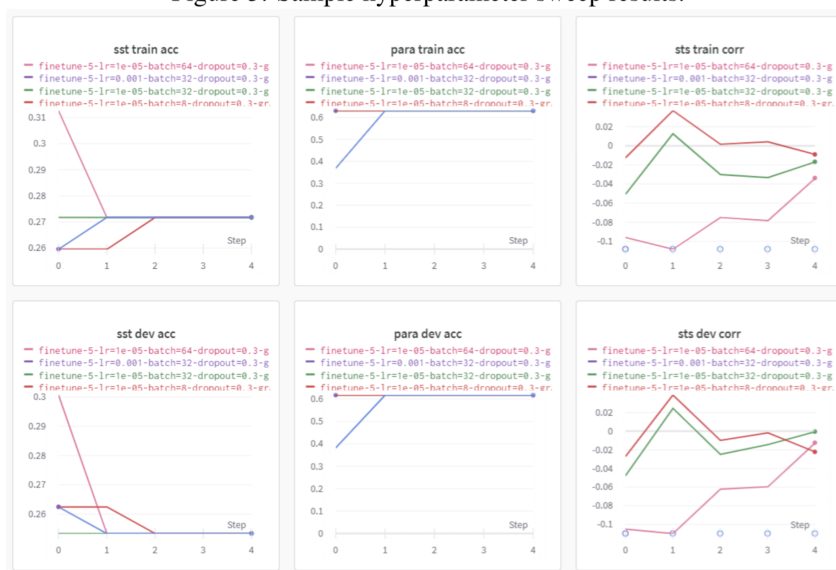
I trained approximately 33 models in total across my hyperparameter sweeps, testing with gradient surgery and mixed-precision training, and my final ensemble model.

[As an aside: it was not until late in my testing that I realized there was a flaw in my STS task head, resulting in poor absolute Pearson correlation despite an upwards trend across training epochs.]

5.4.1 Hyperparameter Sweep

The following image shows the evaluation metrics for each of the three tasks during my random sweep:

Figure 5: Sample hyperparameter sweep results.



Using a high learning rate in Finetune mode led to numerical instability and NaN values, so I ended up settling on running the model in Pretrain mode. The default Pretrain learning rate of 1e-3 worked the best, but a slightly lower dropout probability of 0.2 seemed to work better. Increasing batch size always increased performance across the board from what I saw.

So, the final hyperparameters I settled on were LR=1e-3, dropout probability=0.2, batch size=64, all in Pretrain mode.

5.4.2 Novelties

I performed extended training using the configuration above on 10-20 epochs this time, experimenting with training length, and incorporation of PCGrad and Mixed-Precision Training.

Ultimately, PCGrad had very little impact on the performance of my model on the three tasks between both my own implementation, and the online reference implementation. This is corroborated by other students on EdStem, but still seems strange to me as discussed further below.

Mixed-Precision Training with FP16 led to a fair speedup (15min per epoch to 11min per epoch on Google Colab), but led to two surprising results: (1) model curves were much more unstable, and (2) model accuracy seemed to improve in the long run.

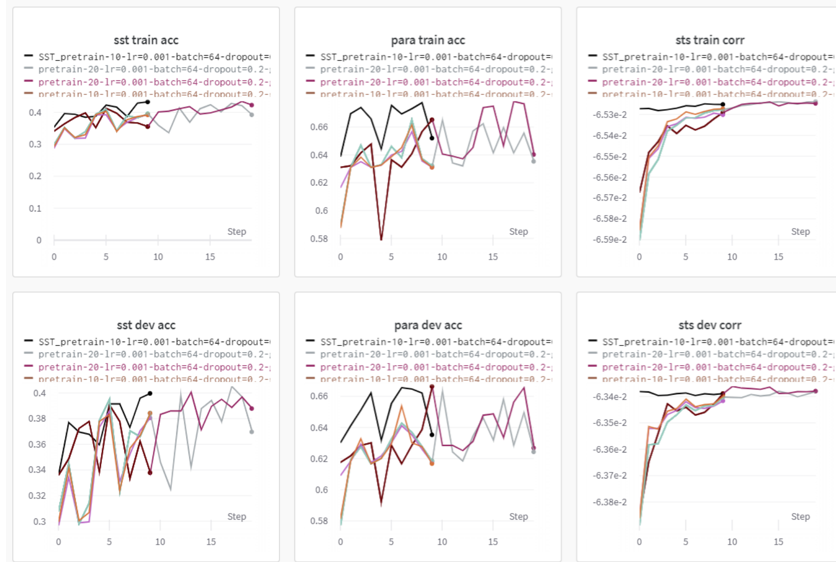
Ultimately, I used my default configuration with Mixed-Precision Training, but no PCGrad.

5.4.3 Ensemble Models

I took my final model above and trained for 20 epochs. I then reloaded the model three separate times, and finetuned each task head specifically on its respective dataset. I then combined the models into a single ensemble for evaluation.

Figure 6: Longer testing with Gradient Surgery and Mixed-Precision Training.

Figure 7: Final ensemble model with fine-tuning.



Above are the final results, trained for 20 epochs. The black curves are the per-task finetuned performance, showing that finetuning along with an ensemble of models produced the best output.

The final metrics I achieved were 40% accuracy on SST, 66.6% accuracy on Paraphrase, and -0.065% on STS.

6 Analysis

I would have thought my models would perform better than they did. The plots above do show that my accuracy or Pearson correlation do increase over time for all tasks, but not to the extent I expected. This is especially true for STS, which I didn't realize was actually performing quite poorly due to the disproportionate y-axis scaling.

I was very surprised that PCGrad had little, if any effect on my model performance. Though there may be other factors at play, I would have expected a bigger impact nonetheless. I was also surprised that Mixed-Precision Training improved model accuracy. Literature suggests that this would improve training time at the cost of precision, but both factors seemed to improve, with the only tradeoff being increased numerical instability (which I attribute to the downcasting to lower precision from FP32 to FP16).

I think it is interesting – humorous even – that the older, more traditional "model ensemble" method still resulted in the best model, and the "gradient surgery" approach actually did very little. However, we must factor in that the ensemble took approximately three times as long to train given that each submodel was finetuned on a specific task.

7 Conclusion

Improving BERT was an enjoyable experience. On top of implementing a transformer model, I really got to dig deep and experiment with improving its performance, incorporating methods such as hyperparameter tuning, gradient surgery, mixed-precision training, and model ensemble approaches. I also gained exposure to a number of tools such as Weights Biases and PyTorch Lightning/Fabric that I am sure to use in my own projects in the future. I think a limiting factor of my model's performance was the amount of complexity in the base model itself (along with the error contributing to such poor STS task performance), along with the amount of experimentation I was able to perform.

With more time, I would have liked to do more experimentation with the fundamental model architecture and experiment with things such as learning rate decay, more sophisticated task heads, different optimizers, and other suggested extensions, but I think I am happy with the amount of experimentation and testing I performed notwithstanding the actual final model performance.

Overall, I am quite happy with the progress made, especially given that I am working on this project alone.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina N. Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2017. Mixed precision training. Cite arxiv:1710.03740Comment: Published as a conference paper at ICLR 2018.

Wei Cheng Tseng. 2021. Pytorch reimplementation for "gradient surgery for multi-task learning".

Kumar S. Gupta A. Levine S. Hausman K. Finn C. Yu, T. 2020. Gradient surgery for multi-task learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.